# Analysis of an Application on Red Storm

**Courtenay T. Vaughan** *and* **Sue P. Goudy**, *Sandia National Laboratories*[1]

**ABSTRACT:** *CTH is a widely used shock hydrodynamics code developed at Sandia. We will investigate scaling on Red Storm to 10000 processors and will use those results to compare with an execution time model of the code.*

**KEYWORDS:** Red Storm, XT3, CTH, performance modeling

## 1. Introduction

In this paper, we will investigate the scaling of CTH on Sandia's Red Storm computer. Red Storm is a CRAY XT3 with 10368 processors that are connected in a 27 x 16 x 24 mesh. The mesh is a torus in the z direction. The processors are 2.0 GHz.

CTH is an explicit, three-dimensional, multimaterial shock hydrodynamics code which has been developed at Sandia for serial and parallel computers. It is designed to model a large variety of two- and three-dimensional problems involving high-speed hydrodynamic flow and the dynamic deformation of solid materials, and includes several equations of state and material strength models [1].

The numerical algorithms used in CTH solve the equations of mass, momentum, and energy in an Eulerian finite difference formulation on a three-dimensional Cartesian mesh. CTH can be used in either a flat mesh mode where the faces of adjacent cells are coincident or in a mode with Automatic Mesh Refinement (AMR) where the mesh can be finer in areas of the problem where there is more activity. We will be using the code in a flat mesh mode for this study.

For this study, we will be using a shaped-charge problem that scales with the number of processors. The shaped-charge consists of a cylindrical container filled with high explosive capped with a copper liner. When the explosive is detonated from the center of the back of the container, the liner collapses and forms a jet. This is illustrated in Figure 1. The problem is run in quarter symmetry, while the image is reflected across the axis to give a full view of the problem. The time 0 image shows the initial state of the problem, while the image at 0.3 ms shows the formation of the jet. The colors in the explosive in the image at 0.3 ms indicate pressure. The simulation consists of those three materials and a fourth material that forms a target for the jet. The target is not shown in Figure 1.

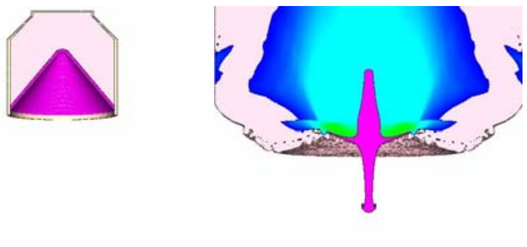

Figure 1: Shaped-charge problem at two times

## 2. Results from Red Storm

We ran CTH with the shaped-charge problem on several numbers of processors up to 10360 processors. The results are shown in Table 1. The time in the table is in seconds.

[1] This research was sponsored by Sandia National Laboratories, Albuquerque, New Mexico 87185 and Livermore, California 94550. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

| Number of Processors | Time per Time Step | % Efficiency |
|---|---|---|
| 1 | 11.83 | 100.0 |
| 2 | 14.23 | 83.1 |
| 4 | 14.86 | 79.6 |
| 8 | 17.17 | 68.9 |
| 16 | 17.49 | 67.6 |
| 32 | 18.70 | 63.2 |
| 64 | 18.86 | 62.7 |
| 128 | 19.73 | 59.9 |
| 256 | 19.86 | 59.6 |
| 512 | 21.95 | 53.9 |
| 1024 | 22.01 | 53.7 |
| 2048 | 22.16 | 53.4 |
| 4096 | 22.10 | 53.5 |
| 8192 | 24.69 | 47.9 |
| 10360 | 22.26 | 53.1 |

Table 1: CTH results from the shaped-charge problem.

The time per time step was determined by taking an average of the time for the first 100 time steps. The problem is scaled so that each processor could have a 90 x 216 x 90 grid of cells. The code may distribute the cells differently since it seeks to make the processor domains as cubic as possible. If the code had scaled perfectly, then the time per time step would remain constant since each processor has the same amount of work. The efficiency is the time on one processor divided by the time on multiple processors and reflects the departure from ideal scaling. These inefficiencies are the result of the necessary interprocessor communications as well as the load imbalance between the processors.

For a flat mesh CTH problem, the problem space is divided into a rectilinear grid of computational cells. The simulation variables, such as pressure and temperature, are stored in three-dimensional arrays and are updated a k-plane at a time. To update these variables may require the values of variables in the 26 neighboring cells, so some subroutines will operate on three k-planes at once.

Each processor's domain is a rectilinear subgrid of this global grid and includes a layer of ghost cells on each face which contains grid information from neighboring processors. The processors are also arranged in a grid so that any two processors that share a face share the whole face and only have each other to exchange that information with. At several points during each time step, each processor takes updated information from its outermost layer of real cells and sends it to neighboring processors which put that information into their ghost cells. Corner cases are handled since a processor only communicates in one direction at a time and communicates the ghost cells that are contained in the plane of real cells that it is exchanging. After an exchange in all three directions, the corner ghost cells will contain data from the processor on the diagonal without communicating directly with that processor. There are also times in the time step, such as determining the size of the next time step, where a global reduction is done.

The number of processors that a processor needs to exchange information with varies with the number of processors in the simulation. Given that the processors are arranged in a grid, each processor can communicate with 0, 1, or 2 processors in each direction. For this simulation, some processors will communicate with 2 processors in each direction starting at 128 processors. After that point the time per time step flattens out.

## 3. A Model of CTH

The model that we are using for CTH timing is taken from [2]. In general, the computational complexity for each time step is $O(N^3)$ where N is the length of one edge of a subdomain assigned to a processor. The communication complexity is $O(N^2) + O(log(P))$, where P is the number of processors used in the simulation. The total run time is the sum of the times for the computational and communication phases since the code does not overlap communication and computation. In equation 1, T represents the run time for a single time step.

$$T = E(\kappa,\varphi)N^3 + C(\lambda + \tau k N^2) + S(\gamma\ log(P)) \qquad (1)$$

In this model, $\lambda$ and $\tau$ denote the communication latency and transfer cost. The ghost cell data exchanges occur in parallel and are point-to-point communications between logical nearest neighbors. The count of exchanges, C, depends on the problem, the dimensionality of the problem, and the number of processors that the simulation is run on. The number of variables in any exchange, $k$, also depends upon the simulation. For this simulation, $k = 40$. The cost of collective operations is O(log(P)). The number of collectives per time step is S and $\gamma$ is the cost of transfer of a double precision number along a leg of the communication tree.

The time for a calculation in a cell is $E(\kappa,\varphi)$, a function of the number of floating-point operations per cell, $\kappa$, and the effective floating-point computational rate, $\varphi$. Since a typical CTH simulation will have regions of high activity as well as cells with differing numbers of materials, the operation counts per cell can vary widely across the domain. The operation count per cell will also depend on the equation of state and response models for the materials in the cell.

## 4. Application of the Model to the Simulation

We have instrumented the code to determine the message types and counts. For this simulation, there are 58 locations where ghost cell data can be exchanged. Since the number of exchanges varies with the number of neighbors that a processor has, some of those exchanges will not occur and the others will exchange data with 1 or 2 other processors. For example, the two processor simulation has 22 exchanges per time step while the 128 processor simulation has 117 exchanges per time step. Given the size of the grid on each processor for this simulation, each processor is sending an average of about 600,000 double precision numbers per exchange.

Every place in the code where the processors could exchange ghost cell data contains a collective operation and there 31 other collective operations for this simulation for a total of 89 collective operations per time step.

We used the Pallas Benchmark to determine the communication parameters. Since CTH uses data exchanges, we used the PingPing benchmark to determine that $\lambda = 8.3\mu s$ and $\tau = 0.00102\ \mu s$/byte or $0.00816\ \mu s$ per double precision number. From the AllReduce benchmark, $\gamma = 10.5\ \mu s$ per double precision reduce.

Given these parameters and the average message size of about 600,000 double precision numbers, each message should take about 4.9 ms to be transmitted. Since the latency for a message is about 8.3 μs, this would seem to indicate that the code should be fairly insensitive to latency.

Using the communication parameters in equation 1, and using the time on one processor for the computational time on multiple processors, the predicted execution time varies from 11.94 seconds on 2 processors to 12.41 seconds on 10360 processors. Clearly, the execution time model does not account for the additional time for running on multiple processors.

One portion of CTH that is not included in this model is the code that prepares messages for communication and distributes the message contents once those are received. As the code goes through processing k-planes, it assembles the messages that need to be exchanged. Only the data for the messages in the z direction is continuous in memory, but even that data gets copied to combine the data for all of the variables that need to be communicated. There is also some amount of computation that is involved in assembling the message to be sent and distributing the data from the message once it is received. CTH also copies the data into a collected MPI datatype, since that is needed for the AMR mode of the code.

Another explanation for the difference in the predicted time versus the actual time is the load imbalance in the problem. Since the shaped-charge problem starts out with a fairly localized area of high activity, once the problem is distributed onto multiple processors, some processors will be more active while others will be less active.

## 5. Comparison with Profiling

We also used CrayPat to profile CTH on several problem sizes to compare with the model. We were not able to profile the entire code due to the current limitations of CrayPat, so we only profiled the MPI calls. We ran each simulation twice, once for 105 time steps and once for 5 time steps and subtracted the quantities for the two runs and divided by 100 to get values for an average time step and to disregard the startup and ending cost for the simulation.

We found that the volume of message traffic is consistent with the number and length of messages predicted in the model. Furthermore, the amount of time reported having been spent in the combination of the MPI_Send and MPI_Recv calls is about a factor of 2 larger than the model predicts. Part of that may be due to difference between the MPI calls that CTH uses versus the MPI calls that the Pallas benchmark uses. There may also be some contention in the mesh since the processors are sending large messages to logical neighbors in the grid of processors which may not be close in the machine. The Pallas benchmark uses only two processors to benchmark the performance.

We also found that the number of collective operations reported in the profiles to be consistent with the model of the code. The time reported for these collective operations is quite a bit more than the model predicts. The time difference is large enough that it would seem to be an indication of the load imbalance present in the problem instead of a difference in how the benchmark measures the time for collective operations and how the code uses those operations.

For example, on 32 processors, the model predicts that the collective operations should take about 4.7 ms per time step, while the profile shows that they take up to about 4.8 seconds per time step. Taking this difference plus the difference between the times for the model and profiling for the exchange operations (about 0.5 seconds) accounts for over 80% of the difference between the predicted time and the actual time. The remaining second of difference is probably due to the extra computations that CTH has to do to support multiple processors. We have seen similar amounts of time being spent in exchanges and collectives with a corresponding difference from the model on other simulations on different numbers of processors.

We also found through the profiling that there were some calls to miscellaneous MPI calls, such as a broadcast each time step to tell the processors if there had been a command from the operator to shutdown the code. These are not in the model, but take little enough time that they do not significantly affect the run time of the code.

## 6. Summary

We have run a weak scaling study using CTH on up to 10360 processors on Red Storm and then modeled the results. While the model was not completely accurate, we were able to learn things about the code through this modeling effort.

Through profiling, we found that we were able to correctly predict the volume and types of communication that occur in the code. We were able to get close to the time required for the data exchange operations, but need to get better understanding of the amount of load imbalance the occurs in the code.

We are planning to repeat this experiment with a problem that exhibits better load balance to see if we are better able to predict the run time. The version of CTH that we were using for this study is a three year old version of the code and we would like to repeat this experiment with a current version of the code since performance enhancements have been put into the code. This will also help us to impact the performance of this widely used code. After that we would also like to work at modeling the code while it is being used in AMR mode, since that is how the code is used much of the time.

## About the Authors

Courtenay Vaughan is a Senior Member of Technical Staff at Sandia National Laboratories. He can be reached at Sandia National Laboratories, P. O. Box 5800, MS 0817, Albuquerque, New Mexico 87185, E-Mail: ctvaugh@sandia.gov. Sue Goudy is a Senior Member of Technical Staff at Sandia National Laboratories. She can be reached at Sandia National Laboratories, P. O. Box 5800, MS 0817, Albuquerque, New Mexico 87185, E-Mail: spgoudy@sandia.gov.

## References

1. E. S. Hertel, Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylor, L. Yarrington, "CTH: A Software Family for Multi-Dimensional Shock Physics Analysis," *Proceedings, 19th International Symposium on Shock Waves* **1**, 274ff (Université de Provence, Provence, France) (1993).

2. S. P. Goudy, Development of a Modeling Methodology for Hybrid Parallelism", PhD Dissertation, NM Tech, May 2005, SAND 2005-2876P.